DOCUMENT RESUME

ED 076 391 SE 015 900

AUTHOR Milner, Stuart
TITLE The Effects of Computer Programming on Performance in Mathematics.
PUB DATE Feb 73
NOTE 41p.; Paper presented at the annual meeting of the American Educational Research Association, New Orleans, Louisiana, February 1973

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Computer Oriented Programs; Computer Programs; *Elementary School Mathematics; Grade 5; *Instruction; Mathematics Education; Problem Solving; *Research

ABSTRACT

The purpose was to investigate the effects of computer programming on performance in mathematics. The LOGO programming language was taught to 18 fifth graders. A pretest-posttest design was employed to determine whether the mathematical concept of variable could be learned through computer programming; a non-computer group of 20 fifth graders was used as a control group. Results indicated that the notion of variable could be learned through computer programming. In addition, three instructional methods for teaching programming were studied. The methods--algorithm-given, incomplete-complete program, no information given--were considered in terms of performance in writing programs. Findings showed that although instructional method may facilitate the learning of programming, there were no significant differences in the criterion situation. Results also showed that there was no effect due to ability (Stanford Achievement Test scores) in the criterion situation. Observation results indicated that the students developed certain problem-solving behaviors and that programming is an effective learning resource in terms of affective considerations. (Author/DT)

FILMED FROM BEST AVAILABLE COPY

U S DEPARTMENT OF HEALTH
EDUCATION & WELFARE
OFFICE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY

ED 076391

THE EFFECTS OF COMPUTER PROGRAMMING ON PERFORMANCE IN MATHEMATICS

Stuart Milner
University of Pittsburgh

The research reported herein was supported by the National Science Foundation and by the Learning Research and Development Center, supported in part as a research and development center by funds from the National Institute of Education. The opinions expressed in this publication do not necessarily reflect the position or policy of the sponsoring agencies and no official endorsement should be inferred.

SE015 900

THE EFFECTS OF COMPUTER PROGRAMMING ON PERFORMANCE IN MATHEMATICS[1]

Stuart Milner
University of Pittsburgh

Computer programming involves the development of algorithms, or unambiguous sequences of operations, for problem solution. In the process, there are a series of steps which include analyzing the problem, devising and implementing a program using the algorithm for solving the problem, testing the validity of the program, and, if necessary, finding errors in or debugging that program. In effect, by specifying a program for problem solution, the student is the teacher and the computer is the student.

As such, programming provides a natural context for the acquisition of basic problem-solving skills. Furthermore, as Feurzeig, Papert, et al. (1969) state, programming provides an excellent context for learning key concepts in mathematics such as variable and function since they can be dealt with in concrete situations. Other support for computer programming in the learning of mathematics is given by Dwyer (1971), Hatfield and Kieren (1972), and the K-13 Arithmetic-Algebra Committee (1971). In the study reported here, programming was taught to fifth grade students to determine its effects on performance in mathematics.

The use of programming as a way of teaching mathematics has been advocated by Feurzeig, Papert, et al. (1969). They state that certain aspects

---

[1]Paper presented at the Annual Meeting of the American Educational Research Association, New Orleans, Louisiana, February, 1973.

of computer programming such as recursion, subroutine, etc. can provide a framework for the learning of mathematics. The precision involved in programming can facilitate the learning of important activities including articulation of mathematical processes, rigorous thinking, and problem solving.

A programming language, LOGO, (Feurzeig, Papert, et al., 1969), was designed specifically to teach mathematics through programming and to: 1) be accessible by young children not familiar with elements of mathematical thinking; 2) facilitate the definition of procedures; and 3) make concepts in mathematics expressible in a natural fashion.

Development of curricula to teach mathematics by programming has occurred in the areas of algebra (Feurzeig, Papert, et al., 1969), number theory and logic (Feurzeig, Lukas, et al., 1971) and problem solving (Feurzeig and Lukas, 1971). Typically, complex programs are written by first writing simple programs, and then generalizing them into more complex ones.

Papert (1971a, 1971b, 1971c) has initiated a project to develop new methods and materials for using computers and computer-controlled devices by elementary school students. He is in favor of letting children "do mathematics rather than merely to learn about it." In doing so, students could conceivably learn rigorous thinking habits and develop heuristic concepts in problem solving. Examples of projects for this are given by Papert and Solomon (1971).

The previous research in this area is very encouraging. It tells us that: 1) programming can provide a framework for learning about mathematics; and 2) elementary mathematics students are capable of learning aspects of mathematical thinking such as the use of heuristics in problem solving and the expression of procedures algorithmically. However, it does not tell us a great deal about how to teach programming. This is important since a student needs to learn the techniques of programming before he can use it as a way of learning about mathematics. In addition, we need to further substantiate the effectiveness of computer programming in learning mathematics.

The present study was designed, in part, to investigate how to teach programming. Are there certain instructional methods that facilitate the learning of programming? Whatever the case may be, if we are to consider the generalizability of teaching programming, it is important to make the method or methods of instruction explicit and replicable. In doing so, it is also necessary to demonstrate the work done by the students themselves as a function of that instruction. It is difficult to determine from the literature of some programming projects, (e.g. Papert, 1971c; Feurzeig, Lukas, et al., 1971), how much work was actually done or could be done by students on an independent basis.

This study was also designed to investigate how computer programming can enhance the learning of mathematics. Specifically, it deals with the question of whether or not the mathematical concept of variable could be learned through programming. Other aspects of mathematics such as the acquisition of general problem-solving skills are also studied.

Method

Design

In this study 18 fifth grade students were taught programming. They were selected on a random basis from the population of fifth grade students at the Oakleaf School, an elementary school in suburban Pittsburgh. These students had no prior experience in computer programming.

The study included three phases of programming activity. The time allotment for programming in each phase was approximately five weeks--two 40 minute sessions per week. The five weeks or ten session period for each phase was a maximum since some students were able to complete the work in a given phase in less time due to the individualized nature of the study.

Phase I dealt with training in the use of the LOGO language. All students were given this training, which consisted of computer-assisted lessons. The lessons, some of which were developed at Bolt, Beranek and Newman, Cambridge, Mass.,[2] dealt with LOGO commands such as those for defining, executing, and terminating procedures, specification of inputs and outputs, and arithmetic operations. The format of a lesson consisted of a brief tutorial followed by a period of independent work in which the student had complete control over the computer in terms of amount and nature of practice. Once the student decided that he had completed the practice, he could access a new lesson. It was assumed that a minimal knowledge of elements of the LOGO language existed once the lessons were completed.

---

[2]The author wishes to express his gratitude to Wallace Feurzeig and George Lukas for providing lessons which served as a basis for the lessons developed for this study.

The rationale for teaching elements of the language independent of instructional method in learning to program was to avoid confounding learning elements of the LOGO language with the writing of algorithms.

In order to investigate the effects of method of teaching computer programming (Phase II), the students were first grouped into two levels of ability--high and low--on the basis of their previous year's scores on the concept, applications, and computation scales of the Stanford Achievement Test. Specifically, the students' scores were ranked, and those above the median constituted the high ability group, whereas those below were placed in the low ability group.

They were then randomly assigned to one of three instructional methods. Examples of the tasks, which were the same for the three methods, and the respective methods are given in Appendix A. One method consisted of an algorithm given in natural language form to be programmed in the LOGO language by the students. The algorithm was based on a task which was also given to the student. In the second method, students were given an incomplete computer program written in the LOGO language. It was necessary for the students to complete the program, which was also based on a task given, and implement it on the computer. In the third instructional method, students were given no information other than the task definition. The

purpose of investigation of the instructional method variable was to determine if method of teaching facilitates programming in the criterion phase (Phase III).

The programs in Phase II involved tasks that required using variables and the generation of arithmetic and geometric sequences. The maximum number of tasks a student could do in this phase was 4. These tasks were related to the programming problems in Phase III.

Presentation of the programming problem, then, depended on which method a student was assigned. Thus, for a given problem, one student was given an algorithm, a second an incomplete LOGO program, and a third was given no information other than the task definition.

At the onset of Phase II, students were given a manual, which described elements of the LOGO language, how to interact with the teletype, and a sample program written in LOGO for generating a simple arithmetic sequence.

Students were given no explicit information in Phase II other than task definitions and instructional method. Occasionally, students reached an impasse and needed, in the researcher's opinion, some type of help. In accordance with some of the principles advocated by George Polya (1957), they were encouraged to look back at every step of the problem, reflect, think about a related problem, and keep trying.

In Phase III, all students were given tasks similar to the ones in the previous phase except that no explicit information was given to them other than task definitions. The kind of assistance described above

given in Phase II also applied here. So that there would be no upper limit on the number of programs that could possibly be written during the 10 session period, more problems were generated than students were able to solve. One student wrote 10 programs.

The tasks in Phase III were, in a sense, criterion measures of the students' ability to write procedures involving variables and sequences given the respective instructional method. An analysis was performed to determine the effects of instructional method in this phase with the dependent variable being number of programs free of errors and the independent variables being methods and ability.

In order to investigate the hypothesis that the concept of variable could be learned through programming, a pretest-posttest design was used. The students in LOGO were one group (N=18), and the remaining students in the fifth grade (N=20)--the non-computer group--were another.

The development of the test on understanding variables began with the definition of variable by this investigator and several mathematics educators. Behavioral objectives were formed based on the definition and translated into test items. After a pilot administration, an item analysis was performed and all negatively discriminating items were discarded. In addition, items with faulty wording were revised. Reliability of the test was computed using the pretest and posttest correlation for the non-computer group, and was found to be .77. Examples of test items are in Appendix B.

Implementation

Programming was done on an interactive basis using a DECsystem-10 time-sharing system. Students worked, typically, in groups of four, and each student had access to a standard teletype device--KSR-33.

The students were taught how to use the editor and file manipulation commands of LOGO on the DEC-10. It might be added that they became quite skillful at using these.

Each student had a folder, in which the manual and previous work were kept. These folders were retrieved by the students prior to each session, and the material in them was used when necessary.

The following two sections deal with the results of the study. In the next section, statistical results dealing with instructional method and ability, and concept acquisition are presented and analyzed. Additional results including observations on student performance and general comments regarding the study are given in the observational results section.

Statistical Results

Instructional Method

Results of performance in Phase II for the 18 students are presented in Table 1. Mean number of error-free programs for the high and low ability groups were 3.33 and 2.33 respectively. Mean scores on instructional method were 2.833, 3.5, and 2.167 for the algorithm-given, incomplete-program, and no-information groups respectively.

Insert Table 1 about here

In Phase III, a randomized block design based on the same grouping by method and ability as in Phase II was used. The number of observations per cell was three.

Results of Phase III performance are presented in Table 2. The hypothesis of equal abilities with $F_{(1,12)} = 1.73$ and $p < .213$ was not rejected. Mean number of programs written for the high and low ability groups were 3.89 and 2.22 respectively (see Table 3). Although there were a greater mean number of programs written by the high ability group in both phases, there was no significant difference due to ability in the criterion situation.

Insert Table 2 about here

In addition, there was no significant instructional method effect in Phase III. The F-ratio with 2 and 2 degrees of freedom was 2.65, and the probability with which the null hypothesis can be rejected was less than .274. Mean number of programs written for the three groups were 1.67, 4.67, and 2.83. Interestingly, the no-information group had a higher mean score than the algorithm-given group even though the reverse was the case in Phase II.

Insert Table 3 about here

The results indicate no significant effect due to method of instruction in programming in the criterion phase. The data seem to indicate that although instructional method may be relevant in learning

to write computer programs (Phase II), it does not seem to be relevant in terms of its transfer value in the criterion phase (Phase III).

Although the cell sizes are small in the ANOVA, and the possibility of error in inference exists, the statistical findings are consistent with the feelings of this investigator having directly observed the students. Therefore, the findings that instructional treatment and ability are insignificant in the criterion situation are supported statistically and observationally although replication is necessary.

There was no significant interaction effect in Phase III, $F_{(2,12)} = .715$, $p < .509$.

Concept Acquisition

It was hypothesized that through computer programming the mathematical concept of variable would be learned. In order to investigate this, a pretest-posttest design was employed. Analysis of covariance was the statistical technique used with the pretest as the covariate and the posttest as the dependent variable.

A test for homogeneity of regression was performed in order to determine whether or not the within-group regression coefficients were equivalent, and that no systematic differences between the groups existed. The resulting $F_{(1,34)} = .061$, $p > .05$ indicated that the regression coefficients were essentially homogeneous.

The analysis of covariance data is summarized in Table 4. The hypothesis that no differences exist between the two groups with respect

to the acquisition of the concept of variable was rejected, $F_{(1,35)} = 7.433$ $p < .01$. Means and standard deviations on the concept test for the computer and non-computer groups are given in Table 5. The mean pretest scores for the computer and non-computer groups were 33.342 and 33.859 respectively, and mean posttest scores for the computer and non-computer groups were 49.10 and 36.10 respectively.

Insert Table 4 about here

Insert Table 5 about here

In addition to and more convincing than the test results, the students demonstrated their knowledge of variables by virtue of the programs written by them. Consider the following procedure to count by any number between any two numbers:

```
TO LIHY /L/ /P/ /I/                    (procedure name and inputs)
10 TEST GREATERP OF /L/ AND /I/        (test)
20 IF TRUE STOP                        (if stopping number exceeded,
                                        procedure ends)
30 PRINT /L/                           (otherwise, print number)
40 MAKE "L" SUM OF /L/ AND /P/         (increment number by P)
50 LIHY /L/ /P/ /I/                    (do procedure again)
END
```

The above program, written by a student, uses three variables. The procedure name is LIHY. The variable L stands for the starting number, P stands for the incrementing number, and I stands for the stopping number.

Discussion

As a whole, the students in the algorithm-given group seemed to

ignore the algorithms in spite of the researcher's suggestions to pay attention to them. Moreover, there was not much variation in response possible if a student did follow the algorithm closely given the nature of an algorithm. It is suggested that this treatment may be used as a possible resource in teaching programming but not as a primary method in doing so.

A greater mean of number of programs were written by students in the incomplete-program group vis-à-vis the other ones. This particular treatment seemed to most facilitate solving the Phase II and Phase III problems although it cannot be considered significantly different from the others in the criterion situation.

Even though one student in the no-information group did well in Phase II, the remaining students did have problems with the third and fourth tasks, which required terminating a procedure after a finite number of iterations. It is felt that some type of information would have facilitated problem solution in Phase II. Further studies are needed to assess the worth of this "discovery" approach.

It is evident from the data that the concept of variable was learned through computer programming. The significant difference on the posttest between the two groups using the analysis of covariance design suggests that computer programming is an effective instructional tool in learning the concept of variable.

Moreover, it is important to consider the use of variables in the programs written by the students in assessing their knowledge of the concept. Not only did the students demonstrate competency at using

variables, but the major programming problems did not involve inability to use variables. In fact, many students were able to verbalize the meaning of variables when explicating their respective programs. Several students even used numerals as variable names to stand for numbers and were able to explain their meaning. Consider the following program written using numerals as variable names and designed to count by any number using two variables:

```
TO COUNT-WITH-INPUTS /1/  /34/          (two inputs)
1Ø PRINT /1/                            (print first number)
2Ø MAKE 1 SUM OF /1/ and /34/           (increment)
3Ø COUNT-WITH-INPUTS /1/  /34/          (do procedure again)
END
```

In the above program, numeral 1 stands for starting number, and numeral 34 stands for incrementing number.

It should be mentioned that the non-computer group received no training in the concept of variable. It may be argued that the significant difference between the two groups was obvious since the computer group received the training. However, the purpose was to establish the fact that the concept could be taught through computer programming. It is beyond the scope of the present study to investigate the relative effectiveness of teaching a concept by means of computer programming vis-à-vis some other method or methods.

Observational Results

Phase I

The students proceeded rapidly through the lessons in Phase I with

no major difficulties. It was expected that by learning elements of the LOGO language independent of learning to write computer programs (Phase II), any possible confounding of the two would be eliminated. This separation seemed valuable. Students working on Phase II tasks occasionally had difficulty with programming logic but appeared to know what LOGO commands could be used. For instance, some students had difficulty placing the IF TRUE and IF FALSE commands in a procedure although they knew how and why to use the commands.

Another purpose of the Phase I lessons was to get students "over the hump," (Dwyer, 1972) into computing. The lessons were to serve as an aid in learning the LOGO language. Unfortunately, some students became dependent on them. During Phase II and III, some students occasionally expressed a desire to work new lessons of the type seen in Phase I. It should also be mentioned that some students did not spend much time in the independent practice part of the lessons. A lesson consisted of a short tutorial followed by a period of practice, in which the student had control over the amount and type of practice.

For most of the students, the LOGO project was their first extensive experience with the computer. Given this, the Phase I lessons were beneficial in several respects. For one, they provided a means of interaction that was brief, concise, tutorial, and controllable by the learner. Major problems and frustrations were relatively non-existent. In addition, the lessons were useful in enhancing typing skills. It might be added that most of the students developed their typing ability

considerably. Moreover, the students gradually became more careful and precise when interacting with the lessons and seemed to reflect more as they proceeded.

Throughout Phase I, many students had difficulty with the concept of inputting numbers to a program. An input to a computer program is a value provided at program execution time for which a previously defined variable name usually exists. Later difficulties by the students with inputs--primarily in Phase III--led me to believe that inputs could have beer emphasized more in this phase.

Finally, an alternative approach to teaching the LOGO language could have consisted of having students write programs from the start and learn elements of the language as programs were written. It is questionable whether or not this would have been more effective in terms of the enhancement of performance in Phase II and III. However, an argument in favor of the method used is that students demonstrated competence at defining and executing procedures--although there were some problems with inputs--and that major difficulties were not due to a lack of knowledge regarding elements of the LOGO language. Moreover, when there are more than four students working concurrently on the computer, a potential management problem can exist. If that problem exists--and this researcher has experienced it with a computer club recently--the self-contained lessons may prove helpful to both the learner and the teacher.

General Comments

One of the most exciting aspects of this study was the apparent

motivation and enthusiasm that continued throughout the four month period. The students' eagerness to work, strong interest, and perseverance prevailed in spite of occasional frustration and inability to solve particular problems. The same enthusiasm and interest continued six weeks after the study in a computer club, which also consisted of members who were not in the study. Students engaged in teaching other students LOGO, developing a baseball game written in LOGO, and defining their own tasks among other projects. That strong interest could prevail from November to June indicates the value of programming at least in terms of affective considerations.

The structure present in this study was particularly expeditious. As tasks were completed, the students looked forward to ensuing ones and demonstrated considerable self-initiative to write programs. In a subsequent experiment with the computer club it was felt that students benefited less from an unstructured situation. Of course, the groups in the club were considerably larger. Nevertheless, students when asked what they would like to do even after suggestions were given would seem to lack direction and initiative to write programs. It remains to be seen how a structured approach compares with an unstructured one.

The development of certain problem-solving behaviors were evident throughout the study. The planning and debugging of programs, willingness to experiment and investigate, and testing of hypotheses were observable regularly. Also, it was not uncommon for a student to claim,

"I've got it now" or to explicate his thoughts about a problem.

Usually, after completing a procedure, students would try their programs with different values. Occasionally, they observed unexpected results. Mary Ann P. wrote a decreasing arithmetic sequence, which had as variables the starting and decrementing values. She wondered what would happen if the starting value was negative. The procedure, which was named after a contemporary rock group, GRAND-FUNK, and the interactions are listed below:

(program to count by subtracting by any number)

```
TO GRAND-FUNK /BEGIN/ /NUMBER/                          (two inputs)
1Ø PRINT /BEGIN/                                        (print number)
2Ø MAKE "BEGIN" DIFFERENCE OF /BEGIN/ AND /NUMBER/      (subtract)
3Ø GRAND-FUNK /BEGIN/ /NUMBER/                          (do procedure over)
END
     (BEGIN stands for starting number)
     (NUMBER stands for decrementing number)
```

(procedure execution)

```
!GRAND-FUNK -4  7
-4
-11
-18
-25
 .
 .
 .
```

She was quite surprised at the result and then questioned what would happen if she used a negative number as the decrementing value. This interaction was as follows:

```
!GRAND-FUNK 5 -1
5
6
7
8
.
.
.
```

Mary Ann was not only surprised, but jubilant. Proudly she demonstrated her "discovery" to everyone around.

When Ruth Ann S. observed a number generated in her geometric sequence that was four lines in length on the teletype, she claimed it was a "googleplex." She told me that she had heard of a googleplex-- $10^{10^{100}}$ --from her teacher, but that now she was observing one. It might be added that students were fascinated with the large numbers often generated by their programs. Herein lies a value in using the computer when introducing large numbers.

Frequently, students worked on problems which apparently served the purpose of augmenting the solution to another problem. Polya (1957) defines these problems as auxiliary problems. Rich H. used auxiliary problems in attempting to write a procedure that would increment by four starting at any number. First, he wrote a procedure that would count by one. He then wrote procedures to count by two and three. Through these successive approximations, he was then able to write the procedure to count by four.

It was interesting to note that in several cases students used a more general computer program to generate a sequence. For instance, Mike H. wrote a procedure that would decrement by any number when all that was needed was one that would decrease by 1. Polya terms this use of a more general case to solve an easier case "inventor's paradox."

Some of the students offered support for their peers on occasion. For instance, Danny noticed that Debbie was dismayed over an unsuccessful attempt to correctly execute a program. He said, "Debbie, you test,

test, test until you get it right!"

Although many of the students needed some encouragement and assistance when they were unable to solve a problem, it was obvious by Phase III that their work was largely independent. By this it is meant that they required little assistance and worked almost entirely on their own.

Frequently, students demonstrated pride in their work. Not only would they proudly show their work to their classmates, but to students in other grades and passers-by as well. One student typified his "mastery" over the computer by calling it his "slave" while the terminal was spewing output.

The students seemed to particularly enjoy watching the terminal spew output. A number of their problems dealt with the generation of infinite sequences, and they would let their respective programs output for long periods of time--sometimes comparing the length of their outputs--before interrupting the programs.

In many cases throughout the study, students "did their own thing." Mike was interested in rock music. Consequently, he named his programs after rock musicians such as CHICAGO, STEPPENWOLF, ALICE COOPER, etc. His delight in doing so, and the degree to which he was "turned on" in general was obvious to numerous observers. Creating unique solutions to problems, and approaching the problems in a wide variety of ways are other ways in which students controlled the learning situation.

Some of the students whose motivation and "ability" was question-

able in the traditional classroom, left no room for doubt when working with the computer. For example, Chris was labelled on the first day of the study by a school staff member as "slow" and unmotivated. However, from the beginning he was clearly one of the most intense, motivated, and perseverant students in the study. It might be added that members of the school staff were surprised but pleased by this.

Also impressive in this study was the determination of the students. Dave several times asked if he could continue when the period ended in order to complete a problem he was very involved in or take it home or stay after school.

In Phase II, the major difficulty for students who could not solve the third and fourth problems was the inability to terminate a procedure after a finite number of iterations. The students in the no-information group had the most trouble. Procedure termination was covered in the lessons. However, in retrospect it is felt that more emphasis could have been placed on procedure termination.

In Phase III, the use of two inputs was a problem for half of the students. Unfortunately, these students were unable to generalize as they were able to write procedures with one input. In several Phase I lessons, two inputs were used. As in the case with procedure termination, it was felt that the use of inputs could possibly have been dealt with to a greater extent. Also, more programming practice in Phase I might have prevented this problem.

Members of the staff at the Oakleaf School expressed their favor,

as a whole, with respect to this study. In personal communications to me several of them remarked that besides the obvious academic value, computer programming was apparently a motivator for "slow" students. In addition, teachers did not mind students missing regularly scheduled classes, although they did expect students to make-up the material, and were pleased with the enthusiasm for computer programming of the students. As a whole, the staff at Oakleaf seemed to strongly accept the project and its implications. The principal stated that "it is my opinion that the children have learned a great deal more than just LOGO during the time they have been involved (in the study)."

Summary and Implications

The present study was designed to investigate the effects of computer programming on performance in mathematics. Toward this end, the LOGO programming language was taught to fifth grade students at the Oakleaf School, a suburban elementary school near Pittsburgh. In order to determine whether or not a conceptual aspect of mathematics could be learned through computer programming, a pretest-posttest design was employed. Results indicated that, in fact, a mathematical concept--the [redacted] notion of variable in this case--could be learned through computer programming.

In addition, a study of three methods of instruction for teaching programming was made. The instructional methods--algorithm-given, incomplete-computer-program, no-information-given--were considered in terms of performance in

writing programs. It was found that although instructional method may facilitate the learning of programming, there were no significant differences in the criterion situation. Results also indicated that there was no effect due to ability--Stanford Achievement Test scores--in the criterion situation.

An important finding was that elementary school students could write computer programs to generate arithmetic and geometric sequences and involving variables. Some of the programs were complex and included the use of logic, recursion, and variables.

Observational results indicated that the students developed certain problem-solving behaviors. These included the planning and debugging of programs, willingness to experiment, and testing of hypotheses.

On the basis of observation, it was also found that programming is an effective learning resource in terms of affective considerations. In writing programs, the students were highly motivated, perseverant, and enthusiastic. Statements by the staff at Oakleaf were highly favorable and supportive of computer programming. The staff as well as outside observers and this investigator were impressed with the intensity and high degree of involvement exhibited by the students. Some of the students whose motivation was questionable in the traditional classroom, as indicated by their teachers, were "turned on" by computer programming.

The students had some minor programming difficulties such as using two or more inputs to a procedure and terminating a procedure after a finite number of iterations. Nevertheless, they demonstrated considerable

skill in defining and executing procedures, using variables, and incrementing values. It is felt that greater emphasis should be placed on the identification of problems made by students in the course of programming as well as the definition of suitable tasks and the preparation for them in subsequent studies.

The educational and psychological implications of the study were numerous. It was evident that the students were actively involved in their learning, initiating and writing programs, making discoveries, and employing heuristic guidelines such as those advocated by Polya (1957) in the process of problem-solving. The question of whether or not these behaviors would emerge due to other modes of instruction with varying degrees of structure is an interesting one. Consider a continuum of instruction in terms of amount of inherent structure. At one end, highly structured, programmed instruction would lie. It is questionable whether or not the behaviors mentioned would be acquired in this case. This same question also holds for highly structured, author-controlled computer-assisted instruction. In the middle of the continuum lies the instructional nature of the present study. More research will be needed to further validate the acquisition of these behaviors. At the other end are unstructured situations such as those in which students define and solve their own problems. Recent experience using the unstructured approach in a computer club, as well as observations of the approach elsewhere, lead this investigator to question its efficacy at least in terms of student initiative to write programs. It remains to

be seen whether or not initiative as well as the other behaviors develop in the unstructured situation.

A definitive statement regarding the transfer of the acquired knowledge is beyond the scope of this study. However, it was obvious that the students were generalizing in the criterion phase (Phase III) based on their training in the prior two phases. In addition, they demonstrated their ability to generalize their knowledge of variable on an independent measure. Gagné (1970) states that transfer is facilitated by "instructional conditions that will stretch students' minds, encourage the generalization of knowledge, and challenge students to solve problems in novel situations." It can only be speculated that the transfer will be facilitated in the future by the instructional conditions of the present study.

Before computer programming is introduced in the classroom, it will be necessary to define its function in terms of what is to be studied through programming, and how programming relates to existing curricula.

The nature of material to be studied will be largely due to the imagination of curriculum developers. Vast possibilities exist for using programming from mathematics to music. The integration of several subject areas is also conceivable.

The relationship of programming to existing curricula is a more subtle issue. Should programming be an integral part of an existing curriculum such as Individually Prescribed Instruction, or should it be an entity by itself? If the former is the case, programming might be used as an adjunct to existing learning packages. If the latter is con-

sidered, programming might serve as a basis for instruction in areas not covered in existing curricula. The implications of the present research seem to favor a combination of the two.

In addition to defining the function of programming in the classroom, it will be important to determine the role of the teacher. In doing so, the degree of involvement of the teacher needs to be considered. This in turn will assist in determining the amount of training in programming that the teachers will need.

Both the function of programming and the role of the teacher will be due, in part, to the degree of self-containment of the material associated with programming. For example, the self-contained lessons were used in this study to assist students in learning elements of the programming language. Although alternative approaches to teaching elements of a language are available, the lessons seemed to facilitate individualization in terms of the inherent nature of the material and teacher assistance particularly when working with groups larger than four.

Finally, computer programming proved to be an exciting way of learning mathematics for the students in this study. That they did learn about mathematics was evident. That programming can be a valuable tool in mathematics instruction is also evident.

References

Dwyer, T. A. Some Principles for the Human Use of Computers in Education. Int. J. Man-Machine Studies, 1971, 3, 219-239.

Dwyer, T. A. Teacher/Student Authored CAI Using the NEWBASIC System. Communications of the ACM, 1972, 15, 1, 21-28.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., Solomon, S., Programming-Languages as a Conceptual Framework for Teaching Mathematics. Project Report No. 1889, Bolt Beranek and Newman, Cambridge, Mass., 1969.

Feurzeig, W. and Lukas, G. Information Processing Models and Computer Aids for Human Performance. Report No. 2187, Bolt Beranek and Newman, Cambridge, Mass., 1971.

Feurzeig, W., Lukas, G., Faflick, P., Grant, R., Lukas, J., Morgan, C., Weiner, W., Wexelblat, P., Programming-Languages as a Conceptual Framework for Teaching Mathematics. Vol. 1-4, Report No. 2165, Bolt Beranek and Newman, Cambridge, Mass., 1971.

Gagne, R. The Conditions of Learning. 2nd edition. N.Y.: Holt, Rinehart, and Winston, Inc., 1970.

Guenther, W. Analysis of Variance. New Jersey: Prentice-Hall, Inc., 1964.

K-13 Arithmetic-Algebra Committee. K-13 Mathematics: Some Non-Geometric Aspects. Part II: Computing, Logic, and Problem-Solving. Curriculum Series #11 OISE, 1971.

Hatfield, L. and Kieren, T. Computer-Assisted Problem Solving in School Mathematics. J. Research in Mathematics Education, 1972, 3 (2), 99-112.

Milner, S. The Effects of Teaching Computer Programming on Performance in Mathematics. Unpublished doctoral dissertation, University of Pittsburgh, 1972.

Papert, S. A Computer Laboratory for Elementary Schools. M.I.T. Artificial Intelligence Laboratory, LOGO Memo No. 1, 1971a.

Papert, S. Teaching Children Thinking. M.I.T., A.I. Lab. LOGO Memo No. 2, 1971b.

Papert, S. Teaching Children to be Mathematicians vs. Teaching About Mathematics. M.I.T., A.I. Lab. LOGO Memo No. 4, 1971c.

Papert, S. and Solomon, C. Twenty Things to do with a Computer. M.I.T. A.I. Lab. LOGO Memo No. 3, 1971.

Polya, G. How to Solve It (Second Edition) Princeton University Press. Princeton, New Jersey, 1957.

APPENDIX A

Appendix A: Programming Tasks

Page

Phase II Tasks by Instructional Method 29

Phase III Tasks 32

(Algorithm Given)

Write a LOGO procedure which makes the computer count by 2. Make the procedure begin with any number, then add 2 to that number, and so on. Make the computer print the numbers.

Here is an example of how the LOGO procedure should work after you have told the computer how to do it:

1
3
5
7
9
11
13
.
.
.

Below is a list of instructions for doing this procedure. You need to make these instructions into LOGO commands so that the computer will understand how to count by 2.

1) Tell the computer the name of the number that is to be input to the procedure. The name can stand for any number to start with.

2) Make the computer print whatever that number is (the one that is input to the procedure). Remember, there is a name that stands for that number.

3) Add 2 to the number that is input. In other words, make the number stand for 2 more than it was when it was input to the procedure.

4) Do the procedure over again. This time the number that is input will be increased by 2 over the last time. (Remember, a procedure can be done over again by telling the computer the name of the input to that procedure.)

(Incomplete Program)

Write a LOGO procedure which counts by 6 and stops at 50. In other words, make the procedure begin with any number (this might be input to the procedure), add 6 to that number, and so on, until the number is greater than 50. Make the computer print these numbers. No number greater than 50 should be printed.

Here is an example of how the LOGO procedure should work after you have told the computer how to do it. The starting number here is 1.

1
7
13
19
25
31
37
43
49

Below is part of a LOGO procedure for counting by 6 up to 50. Fill in the blanks, and type the LOGO procedure into the computer. Then try it to see if you are correct.

```
TO COUNTSIX /B/
10 TEST_____OF _____AND 50
20 IF_____STOP
30 IF_____PRINT /B/
40 MAKE
     NAME:  "B"
     THING:  SUM OF /B/ AND _____
50 _____ /B/
END
```

(No information)

Write a LOGO procedure which makes the computer count by multiplying by 2. Make the procedure begin with any number to start with, multiply it by 2, and so on. Make the computer print the numbers.

Here is an example of how the LOGO procedure should work after you have told the computer how to do it:

1
2
4
8
16
32
64
.
.
.

Write a LOGO procedure which makes the computer count by any number that you input and stops at any number that you input. You can do this by using three inputs to the procedure. The first input might be the number that the procedure begins with. The second input might be the number the procedure counts by. The third input might be the number that the procedure stops at.

In other words, make the procedure begin with any number (the first input), then add the number that the procedure counts by (the second input) to that number, and so on. No numbers greater than the third input should be printed.

Here is an example of how the procedure should work if the first input (the starting number) is 1, the second input (the number the procedure counts by) is 3, and the third input (the stopping number) is 20:

1
4
7
10
13
16
19

Here is another example with the first input 12, the second input 8, and the third input 44:

12
20
28
36
44

Write a LOGO procedure which makes the computer count by subtracting by any number that you input. You can do this by using two inputs. The first input might be the number you start with (it can be any number to begin), and the second input might be the number you subtract by. Make the computer print these numbers.

Here is an example of how the LOGO procedure should look after you have told the computer how to do it. If the first input (the number the procedure begins with) is 50, and the second input is 12 (the number the procedure subtracts by), the procedure should look like this:

```
 50
 38
 26
 14
  2
-10
  .
  .
  .
```

Here is another example. If the first input is 100, and the second input is 5, the procedure should look like this:

```
100
 95
 90
 85
 80
 75
  .
  .
  .
```

APPENDIX B

Appendix B: Examples of Variable Test Items

Page

Items 35

4) The name ABC can stand for any odd number. Which of the following can the name ABC stand for?

a) 3
b) 3 or 5
c) Neither 3 nor 5
d) 8

3 4 POINTS
0 3 POINTS

8)

W → 11
S → 3
Z → 2

(W+S) + (Z+S) = _____

19 4 POINTS
17, 18, 20, 29 } 2 POINTS

9) What numbers other than 21 and 3 can you put in the BOX and TRIANGLE so that they are names for 21 and 3? (Fill in the blanks)

□ + △ = 21 + 3

□ → _____
△ → _____

NUMBERS SHOULD NOT BE 21 AND 3
THEN, IF THEIR SUM IS 24,
GIVE 2 POINTS EACH
IF 20 ≤ THEIR SUM ≤ 30,
GIVE ½ POINT EACH

17) If N can stand for any even number, what number can go into the BOX ? (Fill in the blank.)

N → (□ + 11)

□ → _____

ANY ODD NO. 4 POINTS
ANSWER CAN BE IN BOX OR SPACE

18) G → 4
BR → G
3•BR = _____

12 4 POINTS
7 3 POINTS

21) Use the following information for this question:

VAL → 3
NUM → 9
Z → 10
P → Z
A → VAL
Q → 12

A • Q = _____ 36 1 POINT ; 15 ½ POINT

Z + VAL + A = _____ 16 1 POINT

3 + P = _____ 13 1 POINT ; 30 ½ POINT

P • A = _____ 30 1 POINT ; 13 ½ POINT

24)

N → 13
RS → N
N=RS= _____

13 4 POINTS
N, RS } 1 POINT

Table 1

Mean Error-Free Programs for Phase II

		Instructional Method			
		Algorithm Given	Incomplete Program	No information	
Ability	High	3.33	4.0	2.66	3.33
	Low	2.33	3.0	1.66	2.33
		2.833	3.50	2.167	2.833

Table 2

Phase III Analysis of Variance Results

Source	df	SS	MS	F	P LESS THAN
Ability	1	12.5	12.5	1.73	.213
Method	2	27.45	13.72	2.65	.274
Ability X Method	2	10.33	5.17	.715	.509
Within	12	86.67	7.22		
TOTAL	17	136.94			

Table 3

Mean Error-Free Programs in Phase III

Ability	Algorithm Given	Incomplete Program	No Information	
High	1.67	5.33	4.66	3.89
Low	1.67	4.0	1.0	2.22
	1.67	4.67	2.83	3.05

Instructional Method

Table 4

Analysis of Covariance Results
on Concept Acquisition

Source	df	SS	MS	F	P LESS THAN
Between	1	1726.14	1726.14	7.433	.01
Within	35	8128.37	232.24		
Total	36	9854.51			

Table 5

Variable Test Means and Standard Deviations

		Computer Group N = 18	Non-Computer Group N = 20
Pretest	Mean	33.342	33.859
	S.D.	23.03	21.54
Posttest	Mean	49.10	36.10
	S.D.	26.27	26.01